Journal of Combinatorial Mathematics
and Combinatorial Computing
www.combinatorialpress.com/jcmcc

Reinforcement learning-based state space dimensionality reduction and optimal control strategy design in robot navigation systems

Hao Zhu[1,] ✉

[1] *The Academy for Microelectronics, The Institute of Brain-Inspired Circuits and Systems, and Zhangjiang Fudan International Innovation Center, Shanghai, 200000, China*

ABSTRACT

There are many mature traditional navigation algorithms, but most of them are insufficient in the function of environment perception and understanding, and reinforcement learning can give robots the ability to learn and make decisions. This paper proposes a robot reinforcement learning navigation algorithm and optimal control strategy based on deep reinforcement learning. Firstly, Markov decision modeling for local planning of the robot navigation system is implemented, and then a POMDP belief space dimensionality reduction algorithm based on the NMF update rule is proposed to address the situation of excessive dimensionality and combined with PRM to achieve global reinforcement learning planning. Finally, considering the external information interference problem, a power controller based on the TD3 algorithm is designed to ensure that the robot navigation system can accurately track the signals even under the external interference environment.The position error of the robot under the TD3 controller tends to be close to 0, which is much lower than that of the robot under the PD controller. The experimental results of this paper show that the designed TD3 controller can effectively improve the trajectory tracking accuracy of the robot navigation system and better realize the optimization of the robot tracking control function.

Keywords: Reinforcement learning, belief space dimensionality reduction, Markov decision making, tracking control, robot navigation

1. Introduction

With the rapid development of science and technology and the increasing maturity of artificial intelligence technology, robots are gradually becoming the partners of human life and work. The robot's navigation system is one of the core technologies to realize its autonomous movement and

✉ Corresponding author.
E-mail address: zh240326@163.com (H. Zhu).
Received 15 January 2024; Revised 25 March 2024; Accepted 30 November 2024; Published Online 16 April 2025.
DOI: 10.61091/jcmcc127b-506

localization [1-3]. Robot navigation system refers to the technical means such as sensors and algorithms, so that the robot can autonomously sense the environment and localize its own position, so as to realize accurate navigation and path planning. In today's era of rapid technological development, robot navigation system has become an indispensable part of various scenarios [4-7]. It can be applied to unmanned vehicles, drones, factory automation and other fields to provide convenience for people's production and life. However, as people's requirements for navigation systems become higher and higher, the optimization and positioning accuracy of robot navigation systems have become a critical issue [8-11].

Optimization of robot navigation system refers to the improvement of algorithms or hardware devices to make the robot's navigation ability more efficient and accurate. First, we can analyze it from the perspective of positioning accuracy. Localization accuracy is an important index to measure the performance of robot navigation system. Only with accurate localization can a robot perform effective obstacle avoidance and path planning in complex environments [12-15]. In order to optimize the positioning accuracy of the robot navigation system, various aspects need to be considered, such as the selection and layout of the sensors, the robot's localization mainly relies on the environmental information acquired by the sensors, so the selection and layout of the sensors play a crucial role in the positioning accuracy. Accuracy, stability and applicable field are also issues to be considered [16-19].

In this paper, the robot navigation optimization problem is divided into local reinforcement learning planning and global reinforcement learning planning. The modeling of Markov decision process is carried out in local planning, and the state space, action space and reward function are designed. The belief state space value dimensionality reduction method is also adopted to realize the state space dimensionality reduction, and the NMF update rule is used to update the belief state space, which speeds up the efficiency of dimensionality reduction. In order to reduce the amount of computation, a sampling-based probabilistic road network map algorithm is used. Finally, a reinforcement learning algorithm was utilized to design a robot navigation controller based on the double-delay depth deterministic policy gradient algorithm. Simulation experiments and comprehensive comparison experiments are used to verify the effectiveness and superiority of the method proposed in this paper.

2. Enhanced learning

2.1. Reinforcement of basic concepts of learning

For reinforcement learning, in general it can be divided into value iteration based reinforcement learning algorithms and policy gradient based reinforcement learning algorithms based on their optimization objectives.

In the value iteration-based reinforcement learning algorithm, the intelligent body collects data in the environment and then updates the current value function, which represents an expectation of the reward and the reward that the current strategy will obtain later in a certain state. The intelligent body explores the environment and learns the current state value function $V^{\pi}(s)=\underset{\pi}{E}\left[\sum_{k=0}^{T} r^{k} r_{t+k+1} \mid s_{0}=s\right]$, or state action value function $Q^{\pi}(s,a)=\underset{\pi}{E}\left[\sum_{k=0}^{T} \gamma^{k} r_{t+k+1} \mid s_{0}=s, A_{t}=a\right]$, for the current strategy.

Specifically, the intelligence collects a sequence in the environment and then corrects the estimate of the current value function based on the reward obtained. Monte Carlo (MC) is the most direct estimation method, but the variance of MC's method is relatively large, and the aim needs to collect a complete sequence before the value can be updated.

$$V(s_t)=V(s_t)+\alpha\left(\sum_{k=t}^{T}\gamma^k r_{t+k+1}-V(s_t)\right) \tag{1}$$

Faceted time-dependent (TD) [20] methods are then based on the Bellman equation TD methods use the reward obtained from the execution of the current action in the environment and the value function of the subsequent state to update the current state value function, the variance of estimating the value function using TD's method will be small and the value function can be updated without waiting for the end of the sequence, but since TD's method uses an estimation of the future to fit the current value function, it is therefore a biased method.

$$\begin{aligned} V(s_t) &= E_\pi\left|r_t+\gamma*r_{t+1}+\gamma^2*r_{t+2}+\ldots\right| \\ &= E_\pi\left|r_t+\gamma*\left(r_{t+1}+\gamma*r_{t+2}+\ldots\right)\right| \\ &= E_\pi\left|r_t+\gamma*V(s_{t+1})\right| \\ &= r_t+\gamma*V(s_{t+1}) \end{aligned} \tag{2}$$

$$V(s_t)=V(s_t)+\alpha\left(r_t+\gamma*V(s_{t+1})-V(s_t)\right) \tag{3}$$

The Deep Q network (DQN) [21] algorithm, as well as a series of its improvements, is an algorithm that updates the state-action value function Q^* of the optimal policy, because when we correctly estimate the state-action value function Q^* of the optimal policy, we only need to maximize the state-action value at each step to obtain the optimal policy.The intelligent body in the Deep Q network algorithm selects an action a_t based on the current state s_t and policy π and executes it to get the next state and reward r_t, and then updates the Q^* function.

$$Q^*(s,a)=Q^*(s,a)+\alpha\left(r_t+\max\left(Q^*\left(s_{t+1},a\right)\right)-Q^*(s,a)\right) \tag{4}$$

Algorithms based on value iteration tend to have better data utilization, but they tend to be more unstable and sensitive to hyperparameters because their updates are not directly targeted.

2.2. Reinforcement Learning Algorithms for Policy Gradients

The algorithm for strategy gradient uses gradient ascent to optimize strategy π_θ as a way to maximize its objective functional equation.

$$J(\pi_\theta)=\mathop{\mathrm{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{H}R(s_l,a_t)\right] \tag{5}$$

We derive the objective function:

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &= \nabla_{\theta_\tau}\mathop{E}_{\tau\sim\pi_\theta}[R(\tau)]=\nabla_\theta\int_\tau P(\tau\mid\theta)R(\tau) \\ &= \int_\tau\nabla_\theta P(\tau\mid\theta)R(\tau) \\ &= \int_\tau P(\tau\mid\theta)\nabla_\theta\log P(\tau\mid\theta)R(\tau) \\ &= \mathop{E}_{\tau\sim\pi_\theta}\left[\nabla_\theta\log P(\tau\mid\theta)R(\tau)\right] \end{aligned} \tag{6}$$

Replaces reward $R(\tau)$ for the entire sequence with a cumulative reward $\sum_{t'=t}^{T}r^{t'-t}R\left(s_{t'},a_{t'}\right)$ after the current state.

$$\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{\mathrm{E}}\left[\sum_{t=0}^{T}\nabla_\theta \log P(\tau\mid\theta)\sum_{t'=t}^{T}\gamma^{t'-t}R(s_{t'},a_{t'})\right] \tag{7}$$

Reinforcement learning all goes to increase the probability of that action, which makes updating less efficient. Therefore the strategy gradient method generally replaces subsequent rewards with some dominance function $A(s,a)$.

$$\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{E}\left[\sum_{t=0}^{T}\nabla_\theta \log P(\tau\mid\theta)A(s_t,a_t)\right] \tag{8}$$

Choosing different dominance functions also affects the variance and bias during the algorithm update. Generalized Advantage Estimation (GAE) is currently the most used advantage function in the strategy gradient algorithm, which combines the MC and TD methods to balance the variance and bias of the estimation of the advantage function and make the strategy gradient method update more stable.

$$A(s_t,a_t) = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t-1}\delta_{T-1} \tag{9}$$

The confidence domain policy optimization algorithm (TRPO) uses a confidence domain optimization method to search for the maximum step size of the current update. TRPO converts the objective function into the form of a natural gradient method when solving the optimal policy, which ensures that the KL scatters of the current policy and the previous sampling policy are within a range to select the appropriate step size. The solution complexity is high using search methods, while the Proximal Policy Optimization (PPO) algorithm reduces the complexity of the objective by writing the constraints on the KL scatter in TRPO into the objective function equation.

$$\begin{aligned} J(s,a,\theta_k,\theta) &= \min\left(\frac{\pi_\theta(a\mid s)}{\pi_{\theta_k}(a\mid s)}A^{\pi_{\theta_k}}(s,a), g\left(\varepsilon, A^{\pi_{\theta_k}}(s,a)\right)\right) \\ g(\varepsilon,A) &= \begin{cases}(1+\varepsilon)A & A\geq 0\\ (1-\varepsilon)A & A<0\end{cases} \end{aligned} \tag{10}$$

In order to make the policy gradient algorithm adaptable to asynchronous updating for large-scale reinforcement learning training, it adds the operation of bidirectional cropping on top of PPO, which further improves the robustness and applicability of the PPO algorithm.

$$J^{policy}(\theta) = \hat{\mathrm{E}}_t\left[\max\left(\min\left(r_t(\theta)\hat{A}_t, \mathrm{clip}\left(r_t(\theta),1-\grave{o},1+\grave{o}\right)\hat{A}_t\right), c\hat{A}_t\right)\right] \tag{11}$$

2.3. Large-scale reinforcement learning system

Large-scale reinforcement learning systems typically divide the entire system into two parts, the executor responsible for performing data sampling, and the learner that performs model updating. Distributed Prioritized Experience Queue (A3C) is an early distributed reinforcement learning framework based on the policy gradient algorithm, which uses multiple actuators to sample the environment and compute the inverse gradient, after which the actuator passes the gradient to the learner, which updates the model and then sends the new model parameters to that actuator.A3C has many problems, such as the actuators do not generally have GPU resources.

BatchA2C is an improvement of A3C. In A2C, all the actuators sample data up to a certain amount and then send the data to the learner and stop sampling data at the same time, the learner receives the data and trains the neural network, and then distributes the new network parameters to each actuator, and the actuator receives the new network parameters and then starts to start a new round of data collection, the actuator and the learner are in a synchronized relationship in A2C. The executor and the learner in A2C are a synchronized relationship, which loses the performance of the whole system

to some extent, but there will be some improvement in the training effect of the algorithm compared to A3C.

Unlike A2C, the executor and learner in IMPALA are asynchronous. The executor will keep collecting data in the environment, and the learner will keep training as long as the data in the experience queue reaches a certain amount, while IMPALA introduces a V-Trace technique to mitigate the effect of asynchrony between the learner and the trainer on the learning of the policy.

$$v_s \overset{def}{=} V(x_s) + \sum_{t=s}^{s+n-1} \gamma^{i-s} \left(\prod_{i=s}^{t-1} c_i \right) \delta_t V \tag{12}$$

Among them:

$$\begin{aligned} \delta_t V &\overset{def}{=} \rho_t \left(r_t + \gamma V(x_{t+1}) - V(x_t) \right) \\ \rho_t &\overset{def}{=} \min\left(\bar{\rho}, \frac{\pi(a_t \mid x_t)}{\mu(a_t \mid x_t)} \right) \\ c_i &\overset{def}{=} \min\left(\bar{c}, \frac{\pi(a_i \mid x_i)}{\mu(a_i \mid x_i)} \right) \end{aligned} \tag{13}$$

Since in the IMPALA and A2C architectures, it is still up to the executor to perform forward reasoning about the network when collecting data, and the executor does not generally hold GPU resources, the speed of data collection by the executor is still not high enough. Later, distributed PPO algorithms go a step further and use a reasoner to specialize in reasoning about the network, and the reasoner generally holds GPU resources together with the learner. In these architectures, the executor no longer holds a copy of the network, but is only responsible for the data computation of the environment. After each step of the environment execution, the executor sends the state generated by the environment to the reasoner, which collects a certain number of states and then focuses on forward reasoning, and then sends the resulting actions to the executor for execution, which collects a certain amount of data and then packages it to the learner for training.

If an IMPALA-like structure is used, the environment can get the action inputs within a certain time, and at the same time there is a certain tolerance for delays and losses in the network because the model parameters are passed between the actuators and the learners. Therefore it is also necessary to choose the appropriate large-scale reinforcement learning architecture according to the actual problem.

3. Robot hierarchical enhanced learning navigation

3.1. Localized intensive learning planning

3.1.1. State space downscaling design. The state space for robot navigation consists of two main parts, the inputs from external sensors and the current motion state of the robot.

The final state of the navigation task $S = [L, \rho, \theta, v_l, v_a]^T$, when taking $6°$ as the laser interval, the state space is a 64-dimensional vector. Since the dimensionality is too high, the state space in the navigation system is downscaled.

In most real-world models, analyzing the internal structure of state variables reveals that they can be represented by a set of random variables, an internal property known as the decomposable property. Decomposable state variables are represented by a random finite set of variables $X = \{X_1, X_2, \cdots, X_n\}$, with each X_i representing a characteristic of the state variable. A state can be represented as $s = \{X_1 = x_1, X_2 = x_2, \cdots, X_n = x_n\}$. Similarly, observation and action variables can be described by decomposability. After the state, observation, and action variables are described using

decomposable descriptions, the POMDP model is transformed into a FPOMDP model, whose state transfer function, observation function, and payoff function can be compressed and represented using a Dynamic Bayesian Network (DBN).Each node in the DBN corresponds to a conditional probability table describing the influence of its father node on the node, with conditional probability distributions of $P\left(X_i \mid parents\left(X_i\right)\right)$. Utilizing the conditional independence, the DBN decomposes the joint probability into the product of several conditional probabilities, i.e. $P\left(X_1,\cdots,X_n\right)=\prod_i P\left(X_i \mid parents\left(X_i\right)\right)$.

Before the variables and transfer functions S, A, Z, T, O, and R are decomposable, the father nodes of the variables at moment $t+1$ are all the variables at moment t, and assuming that there are n variables in total and they are all Boolean, the size of the parameter that needs to be traversed for each action is $O\left(n2^n\right)$. If the variables are represented using a decomposable representation, and assuming that the number of the father nodes of all the variables is k, and $k \square n$, the size of the parameter that needs to be traversed is $O\left(n2^k\right)$. If the variables are represented using a decision tree or an algebraic decision graph using context-independent relationships, where the number of leaf nodes on the decision tree or algebraic decision graph does not exceed $f(k)$, and $f(k)$ represents a polynomial function of the number of nodes in the variable's father, then the size of the parameters to be traversed is $O(n \cdot f(k))$.

In addition, in practical numerical calculations, the probability of zero is taken to be negligible. When computing belief states, cropping out variables with zero probability can reduce the dimension of the belief state space. For a given belief state b, the possible states $\omega(b)$ can be described generically as:

$$\omega(b)=\left\{\left\{x_i\right\}_{i=1}^n \mid \left(\forall x_i\right) P\left(X_i=x_i \mid b\right)\neq 0\right\} \tag{14}$$

$\omega(b)$ is all the "meaningful" states of the intelligent body, in the worst case, $\omega(b)=S$, but the actual situation is $\left|\omega(b)\right| \square \left|S\right|$, so the number of states can be reduced, thus reducing the size of the belief state space.

According to the current state $s \in \omega(b)$, the next state $s' \in \omega'(b)$ can be calculated by using Eq. (15), and $\omega'(b)$ is the set of states at the next moment.

$$\begin{aligned}\omega'(b)=\Big\{s' \mid (\forall s \in \omega(b))\left(T\left(s,a,s'\right)\neq 0\right)\\ \wedge\left(O\left(s',a,z\right)\neq 0\right)\Big\}\end{aligned} \tag{15}$$

Utilizing $\omega(b)$ and $\omega'(b)$, the probability of observing a variable can also be represented compressed by equation (16).

$$P(z \mid a,b)=\sum_{s' \in \omega'(b)} O\left(s',a,z\right) \sum_{s \in \omega(b)} T\left(s,a,s'\right) b(s) \tag{16}$$

Using the above results, the compression process for the belief state update function is:

$$b'_{fc}\left(s'\right)=\eta O\left(s',a,z\right) \sum_{s \in \omega(b)} T\left(s,a,s'\right) b(s) \tag{17}$$

where the subscript fc denotes the belief state after decomposable compression.

In the KL scatter function, the error between B and $F\tilde{B}$ needs to be evaluated, and it is assumed that each approximate solution leads to an ε error which increases the distance between B and $F\tilde{B}$. Then the error accumulated up to time t is

$$\varepsilon+(1-\gamma)\varepsilon+\cdots+(1-\gamma)^{t-1}\varepsilon \leq \varepsilon/\gamma \tag{18}$$

Combining VDC and NMF update rules, this paper proposes a VDC-NMF dimensionality reduction algorithm.

3.1.2. Action Space Design. The motion space of the navigation task $A=\left[v_l', v_a'\right]$, denotes the required target linear velocity v_l' and the target angular human moving forward, while a negative number indicates that the robot is moving backward; $v_a' \in \left[-V_{\max\_a}, V_{\max\_a}\right]$, $V_{\max\_a}$ denotes the maximum angular velocity, when v_a' is positive it indicates that the robot is steering counterclockwise, while a negative number indicates that the robot is steering clockwise.

3.1.3. Reward function design. Human design of rewards is an important part of many reinforcement learning tasks. In the navigation problem, the following reward function is used in conjunction with human intuition about the navigation problem, based on the principle of simplifying it as much as possible:

$$reward\left(P_{now}, P_{goal}\right)=\begin{cases} R_s & success \\ R_m * \left\|P_{now}-P_{goal}\right\|_2 & move \\ R_f & fail \end{cases} \tag{19}$$

P_{now} denotes the current position of the robot, P_{goal} denotes the end point position, $\left\|P_{now}-P_{goal}\right\|_2$ denotes the Euclidean distance between the current position and the end point, R_s denotes the reward after the robot reaches the end point, for R_m denotes the time penalty parameter after each movement of the robot, and R_f denotes the penalty after the robot fails to navigate.

3.1.4. Deep deterministic policy gradient algorithm. In this section, the deterministic policy gradient algorithm based on the framework of Actor-Critic algorithm [22] and its improvement to obtain the deep deterministic policy gradient algorithm will be introduced.

Deep Deterministic Policy Gradient (DDPG) algorithm is a combination of DPG algorithm and deep learning.DPG algorithm has Actor-Critic structure.The main improvement of DDPG algorithm is to represent the Actor and Critic part of it with deep neural network.Actor outputs the policy, Critic estimates the state action value and optimizes it using TD error. Also the gradient is computed for Actor update and all these operations are able to be realized on neural networks. Using deep neural networks, it is possible to handle higher dimensional state spaces as well as perform more powerful nonlinear mappings, thus improving the performance of the reinforcement learning algorithm.

The DDPG algorithm has four networks, Actor-eval network, Critic-eval network, Actor-target network, and Critic-target network.

The Actor-eval network performs the update of the policy network parameters and outputs the action $a_t=\pi_\theta\left(s_t\right)$ used for the interaction based on the current moment state s_t of the environment, and the Actor-target network estimates the action $a_{t+1}=\pi_{\theta'}\left(s_{t+1}\right)$ to be chosen at the next moment based on the next moment state s_{t+1} returned from the interaction, which is used to compute the target Q value. According to the idea of deterministic policy gradient, the goal of the Actor is to maximize the Q value that can be obtained by taking an action in each state with a loss function:

$$J(\theta)=-\frac{1}{m}\sum_{t=1}^{m} Q_\omega\left(s_t, a_t\right) \tag{20}$$

Q_ω that is the output of the Critic-eval network, the Critic-eval network is used to compute the Q value $Q_\omega(s_t, a_t)$ of the current state action, and the Critic-target network is used to compute the target Q value, which is the sum of the discount between the actual reward at the current moment and the value of the state action Q at the next moment $y_t = r_t + \gamma Q_{\omega'}(s_{t+1}, a_{t+1})$. The Critic-eval network carries out the updating of the parameters of the value network, and to approximate the state action value function, the mean square error between the current Q value and the target Q value is used as the loss function.

$$J(\omega) = \frac{1}{m}\sum_{t=1}^{m}\left(y_t - Q_\omega(s_t, a_t)\right)^2 \tag{21}$$

The parameters of the Actor-target network and Critic-target network are obtained from the Actor-eval network and the Critic-eval network, and the parameter tuning method in DQN can be used to completely copy the parameters from the two target networks every certain number of training steps, or a soft update can be performed at every step.

$$\begin{aligned}\theta' &= \tau\theta' + (1-\tau)\theta \\ \omega' &= \tau\omega' + (1-\tau)\omega\end{aligned} \tag{22}$$

τ is the update coefficient that takes a relatively small value.

Compared to the PG algorithm, the efficiency of the Actor-Critic algorithm is somewhat improved by estimating the merits of the next step by Critic, which can be done by updating once at each step instead of collecting a whole round of data and then updating again, however, this still doesn't change the fact that the results of the interactions at each step are learned only once. The experience playback method means that the quaternions $e = \langle s_t, a_t, r_t, s_{t+1} \rangle$ obtained from each step of interaction are deposited into a fixed-size experience buffer pool $E = \{e_1, e_2, \ldots, e_m\}$, and at each step of the learning process a random number of batch-size data are taken out from the experience buffer pool and used for neural network training.

The DDPG algorithm uses the Ornstein-Uhlenbeck (OU) stochastic process to induce random noise during the sampling process, which is given by:

$$dx_t = \theta(\mu - x_t)dt + \sigma dW \tag{23}$$

in discrete space is of the form:

$$x_{t+\Delta t} - x_t = \theta(\mu - x_{t-1})\Delta t + \sigma\Delta W \tag{24}$$

x_t is the generated noise value, μ denotes its mean value, θ and σ are the weight parameters, W is the Wiener process, ΔW is the increment over a time interval, and $\Delta W \sim N(0, \Delta t)$.

3.1.5. Network structure. Although neural networks are powerful in fitting, their computational power increases rapidly as the number of network layers deepens and the number of neurons per layer increases.

Using the deep deterministic policy gradient algorithm, the Actor network for robot navigation loses for state s, and the output is the 2-dimensional action a, and the Critic network loses for state s and action a, and the output is the fitting result for $Q(s, a)$. Since the loses are all of low dimensionality, and in order to be deployed on inexpensive hardware devices, a fully connected network structure with fewer layers is used, this paper designs four network structures on top of the above mentioned reinforcement learning algorithm on which four network structures are designed.Actor network uses fully connected structure.

3.2. Global reinforcement learning-PRM planning

3.2.1. PRM. The establishment of occupancy raster maps of the environment is essentially the transformation of continuous space into discrete space, so that the path between the starting point and the end point can be searched in the map using a graph-based search algorithm. Probabilistic road network map algorithm PRM through random sampling, the same discrete space transformation, can effectively solve the large space and complex conditions of the path planning problem.PRM algorithm is divided into learning and querying of two parts, the learning stage for road network map modeling, that is, given the map, in the space of the random sampling and k nearest neighbor linkage. The query, on the other hand, gives the path based on the given start and end points, which is done by putting the start and end points into the road network map, searching for the shortest path using a graph-based algorithm, and letting the robot move along the shortest path, and the movement between two neighboring points on the path is considered as one local planning.

3.2.2 Road network linkage based on value functions. In the DDPG algorithm, a Critic network is fitted to the state action value function $Q(s,a)$, and the meaning of the Q value is the cumulative discounted reward obtained by taking action a in the current state s:

$$Q_\pi(s,a)=E_\pi\left[\sum_{t=0}^{+\infty}\gamma^t r_{t+1}\mid s_0=s,a_0=a\right] \tag{25}$$

Regardless of success or failure, the robot navigation task ends in a determined amount of time, so the actual cumulative discount reward is not infinitely cumulative, with a value of Q:

$$Q_\pi(s,a)=E_\pi\left[\sum_{t=0}^{n-1}\gamma^t r_{t+1}\mid s_0=s,a_0=a\right] \tag{26}$$

n denotes the number of steps to run, local navigation targets closer targets, so it does not take a large value. With q' as the end point, the larger the action state value of state s_q at position q, the higher the probability that q to q' is reachable.

Design the value function linkage method for determining reachability in the PRM construction algorithm: for point pair (q,q'), randomly sample m bit poses at position q and construct the state s_{qi}, compute the action $a_{qi}=\pi_\theta\left(s_{qi}\right)$ using the strategy, compute the average linkage weight $P(q,q')$, set the linkage threshold P_{th}, and consider q to q' reachable if $P(q,q')\le P_{th}$, and unreachable otherwise.

$$P(q,q')=\frac{1}{m}\sum_{i=1}^{m}Q\left(s_{qi},a_{qi}\right) \tag{27}$$

4. Dynamics controller design based on TD3 algorithm

In this section, deep reinforcement learning techniques are utilized to design a dynamics controller based on the double-delay deep deterministic policy gradient algorithm (TD3), which generates control quantities according to the current navigation state of the robot navigation, continuously reduces the error between the actual navigation heading angle and the desired doodle angle of the robot navigation, and enables the navigation speed to reach the desired value, which in turn realizes the accurate tracking of the desired path.

4.1. Markov Decision Process

The interaction of the robot navigation with the environment constitutes the basic working principle of reinforcement learning: at moment t, the robot navigation acquires state s_t and reward r_t from the environment and performs a new action based on the set of actions a_t. At moment $t+1$, the robot navigation acquires a new state s_{t+1} as well as a new reward r_{t+1} and performs a new action. This cycle continues until the robot navigation finds the optimal strategy, which maximizes the cumulative reward at this moment.

Markov Decision Processes (MDPs) are an important foundation for reinforcement learning and are the implementation of robot navigation to interact with the environment.

The set of state spaces in the decision making process is denoted by S, and s_t denotes the state of the robot navigation at the t moment, denoted as $S=\{s_1, s_2 \ldots s_t\}$.

The set of action spaces in the decision making process is denoted by A. The actions in different environments form the action space A, and a_t denotes the action of the robot navigation at the moment t, $A=\{a_1, a_2 \ldots a_t\}$.

The state transfer probability is denoted by P. The reward function R is the value of the reward obtained by the robot navigation after performing the action a_t in the current state s_t while interacting with the environment.

The discount factor for calculating the cumulative reward is denoted by γ, which satisfies $0 \le \gamma \le 1$.

The strategy defines the action that should be taken by the robot navigation in a certain state, and the strategy is denoted by the symbol π. Knowing the current state, a probability is obtained by substituting the current state into the strategy function:

$$\pi(a \mid s) = p\left(a_t = a \mid s_t = s\right) \tag{28}$$

Under strategy π, the goodness of the corresponding state or action is evaluated by summing the product of the return value and the discount factor, and the goal is to find an optimal strategy to maximize the cumulative return G_t, which can be expressed as:

$$G_t = R_t + \gamma R_{t+1} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k} \tag{29}$$

In order to evaluate the strategy as good or bad, the cumulative reward brought by using strategy π from state s is the value function of the action under the strategy in this paper, which is denoted as:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\left[R_t \mid s_t\right] \\ &= E_{\pi}\left[\sum_{k=1}^{\infty} \gamma^k r_{r+k+1} \mid s_t\right] \end{aligned} \tag{30}$$

The cumulative reward resulting from the robot navigating from state s, taking action a and then using strategy π is called the state value function, which is denoted as:

$$\begin{aligned} q_{\pi}(s,a) &= E_{\pi}\left[R_{t+1} + \gamma R_{t+2} + \ldots \mid s_t = s, a_t = a\right] \\ &= E_{\pi}\left[G_t \mid s_t = s, a_t = a\right] \end{aligned} \tag{31}$$

The state value function and the action value function are split into the immediate reward and the discounted value of the subsequent state, with the action value function having the action already determined at moment s, and the state value function action executed based on a probability distribution. The Bellman expectation equation based on is:

$$\begin{aligned} v(s) &= E\left[G_t \mid s_t = s\right] \\ &= E\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid s_t = s\right] \\ &= E\left[R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid s_t = s\right] \\ &= E\left[R_{t+1} + \gamma G_{t+1} \mid s_t = s\right] \end{aligned} \tag{32}$$

4.2. TD3 reinforcement learning algorithm

The TD3 algorithm is a further improvement of the deep deterministic gradient algorithm (DDPG), which effectively improves the Q-value over-estimation problem of DDPG, which includes key techniques such as truncated Double-Qlearning, delayed target updating, and target policy smoothing.

First, for truncated Double-Qlearning, the Q-value overestimation problem often exists in Deep Q Networks (DQNs) and DDPGs, and TD3 introduces a DoubleQ-Learning-like way to update the Cirtic network in DDPGs, i.e., building two Q-value networks and taking the minimum of them to avoid the overestimation problem, then there is:

$$y\left(r, s'\right) = r + \gamma \min_{i=1,2} Q_{\theta_1'}\left(s', \pi_{\phi_1}\left(s'\right)\right) \tag{33}$$

where $y\left(r, s'\right)$ denotes the final Q-valued function, $Q_{\theta_1'}$ and $Q_{\theta_2'}$ denote the values produced by the two Q-valued networks. r denotes the current reward, and γ denotes the reward discount factor.

After calculating the final Q-value function, the mean square errors of the two Q-values with respect to the target are calculated separately, and these two errors are summed to obtain the loss function Critic Loss of the network, which is given:

$$\begin{cases} L\left(\phi_1, \mathrm{D}\right) = E\left[\left(Q_1(s,a) - y\left(r, s'\right)\right)^2\right] \\ L\left(\phi_2, \mathrm{D}\right) = E\left[\left(Q_2(s,a) - y\left(r, s'\right)\right)^2\right] \\ CriticLoss = L\left(\phi_1, \mathrm{D}\right) + L\left(\phi_2, \mathrm{D}\right) \end{cases} \tag{34}$$

where, $L\left(\phi_1, \mathrm{D}\right)$ and $L\left(\phi_2, \mathrm{D}\right)$ are the mean square error, D is the empirical playback for storing the historical data for easy batch updating, $Q_1(s,a)$ and $Q_2(s,a)$ are the Q-values of two Q-valued networks, and ϕ_1 and ϕ_2 are the parameters of two Actor networks.

For the target strategy smoothing, considering the peak of DDPG in the value space there may be overfitting phenomenon, which will lead to the strategy to be affected by the peak too much, so the target strategy smoothing technique is proposed to add noise as regularization in the generation of the target strategy, which can smooth the computation of the Q values to avoid overfitting, then there is:

$$y\left(r, s'\right) = r + \gamma Q_{\theta'}\left(s', \pi_{\phi'}\left(s'\right) + \right), \sim clip\left(N(0, \sigma), -c, c\right) \tag{35}$$

where ò denotes the truncated noise, obeying a normal distribution. Adding interference to the input actions of the target network is equivalent to fuzzy fitting the values of a small region around the target action.

The TD3 reinforcement learning flowchart is shown in Fig. 1. In the training phase, a small batch of samples $\left(s, s', a, r\right)$ is uniformly sampled from the empirical playback pool, followed by inputting the observations s' into the target Actor network to get the next moment action a' and inputting $\left(s', a'\right)$ into the two target Critic networks to get the two Q values and compute the lesser of their two values, using them to compute the $y\left(r, s'\right)$ and the mean-square error, and then back-

propagating to update the Critic networks. Next, the Actor network is updated every d time using the Q values computed from the first Critic network in the main network. Finally, the parameters related to the target network are updated by means of soft updates.

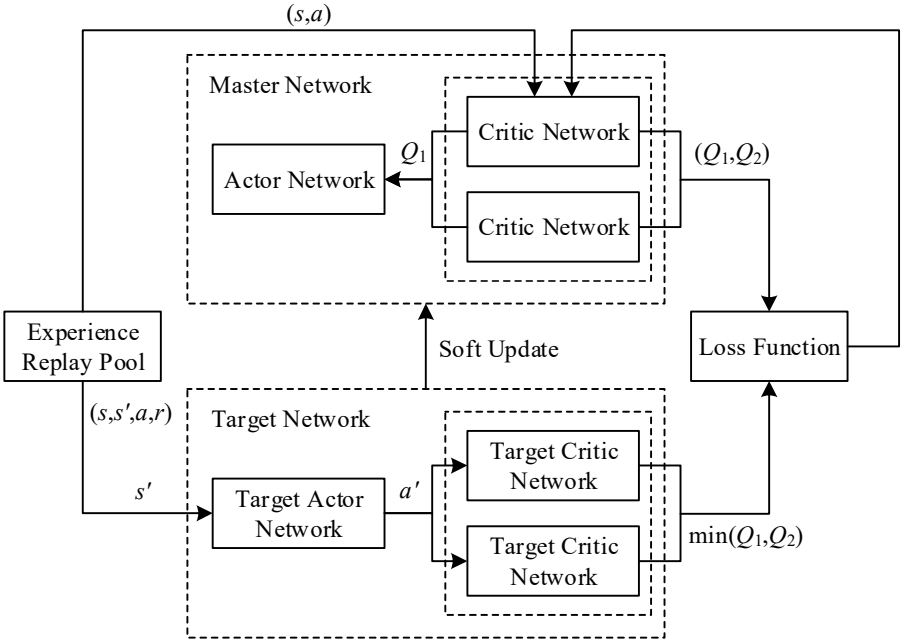

Fig. 1. TD3 reinforcement learning flowchart

4.3. State space, action space and reward function design

Markov modeling is indispensable in analyzing reinforcement learning problems, and its model design is mainly divided into three parts, which are the design of state space, action space, and reward function. In this chapter, based on the robot navigation path tracking problem, the Markov model is defined as follows:

1) State space. In this paper, the desired path is navigated by line-of-sight guidance method. The action value of the previous moment is input into the state space, which can get more accurate results by comparing the two moments before and after, and improve the input jitter problem.

In summary, the state space of the current moment t can be designed as:

$$s_t = \left(x_e, y_e, \psi, \psi_e, u, v, r, u_e, \tau_{u(t-1)}, \tau_{r(t-1)}\right) \tag{36}$$

2) Action space. In this paper, we mainly study the path tracking problem in 2D plane, combining the line-of-sight guidance method and deep reinforcement learning algorithm to track the desired path by continuously reducing the error between the actual bowing angle and the guidance angle. According to the control flow designed in the previous chapters, if the underdriven robot navigates in the horizontal plane for the control, it needs the longitudinal thrust force F_u and the bowing torque F_r to complete the tracking task, so the action space of this paper is designed as follows space is designed as follows:

$$a = \left(F_u, F_r\right) \tag{37}$$

3) Reward function. The purpose of the reward function is to motivate the robot to navigate and explore the environment continuously to accumulate more rewards, if the reward function is not set reasonably, it will lead to the phenomenon that the robot navigates in the same place or acts

haphazardly. Combined with the path tracking control objective, considering the relative distance, relative speed and relative bow direction, the first part of the reward function r_i is designed:

$$r_1 = \lambda\left(2\exp^{-k_1|u_e|}-1\right)+\left(2\exp^{-k_2|\psi_e|}-1\right)+\left(2\exp^{-k_3\left|\sqrt{x_e^2+y_e^2}\right|}-1\right) \tag{38}$$

The reward value is calculated in the way of $2e^{-|input|}-1$ because it can limit the size of the reward value between $(-1,1)$ to prevent the occurrence of too high reward value.

The reward function is designed by calculating the standard deviation value in a certain number of step cycles to solve the problem of robot navigation control input jitter, so the second part of the reward function r_2 is designed as:

$$r_2 = \exp^{-k_4|\sigma_{t_u}|}+\exp^{-k_4|\sigma_{t_v}|}-1 \tag{39}$$

Here, σ_u and σ_r are the standard deviations of the longitudinal thrust and the bow torque of the controller output.

In summary, the expression of the reward function for path tracking control is:

$$r = r_1 + wr_2 \tag{40}$$

5. Comparative experimental analysis of the performance of model network structures

This section builds a baseline experimental model that serves as a baseline for algorithm comparison. Most of the experimental parameters in the later subsections are consistent with those of the baseline experimental model. The experiments in this section are conducted in Scene 2 with a laptop platform and 1000 training acts.

The epsilon parameter chosen for the Epsilon greedy action starts at 1, with a decline rate of 0.96 per round, and Figure 2 shows the variation of the epsilon parameter with the number of training acts.

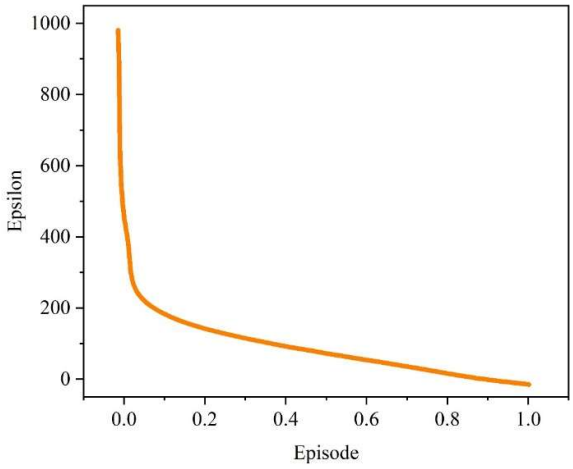

Fig. 2. Epsilon changes with the number of training days

Figures 3, 4, and 5 show the baseline model's average score per act, success rate, and average steps per act as a function of the number of training acts for Scene 2, respectively (the data is averaged every 50 acts on a sliding scale). The baseline model's success rate reaches 80% for the first time in Act 598, reaches 90% in Act 630, and stays above 80% between Act 598 and Act 1000. The average number of steps rises and then falls, reaching a maximum of over 90% around act 200, when most training rounds end with a timeout. Then the average number of steps gradually declined, and the success rate and

score increased, stabilizing around act 600, and after the number of steps stabilized, the success rate of the model also stabilized. The reason for the average number of steps to rise and then fall is that the initial positive sample is too small, and AGENT chooses a conservative turn-in-place strategy in order to avoid collisions during the learning process.

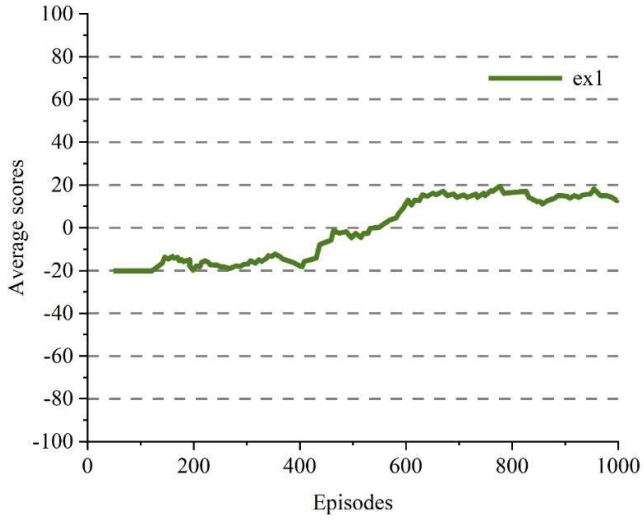

Fig. 3. Average score

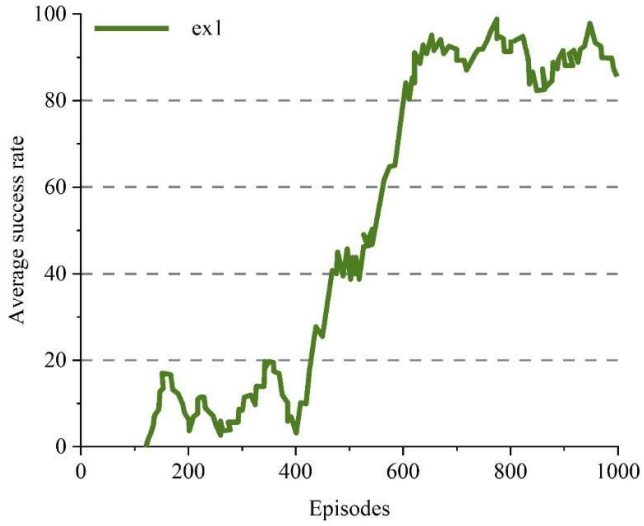

Fig. 4. Average success rate

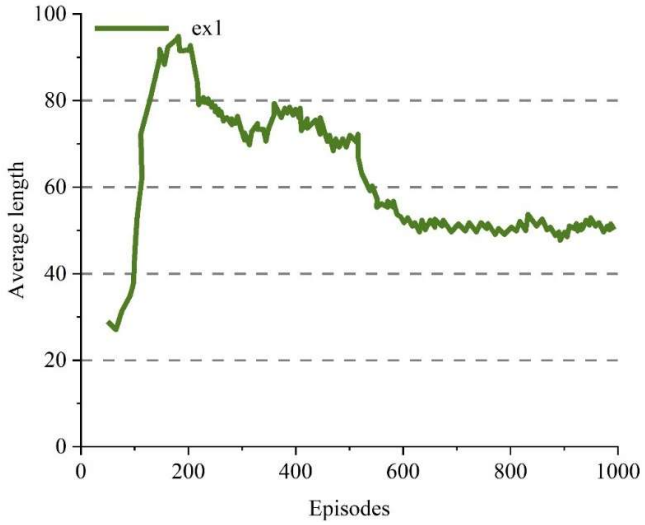

Fig. 5. Average length

In this section, the Double DQN algorithm, two network layers (4 layers/3 layers), and two network structures of DDPG and TD3 are used for comparison experiments. The experimental scenario is Scene 1, there are 4 experiments, the experiment number and network structure are shown in Table 1, keeping the other experimental parameters the same, the number of training acts for each experiment is 300, and the experiments are conducted on a laptop platform.The 4 experiments are carried out under Scene 1 with the training, and the changes of the average scores and the success rates with the number of training acts are shown in Fig. 6 and Fig. 7, respectively, and the data are subjected to a sliding average of 50 acts. Under Scene 1 of the simple task, the results of the experiments with different network structures are almost indistinguishable, and all of them reach a success rate of 62% around 155 acts. There may be two reasons for analyzing this phenomenon: first, the input dimension is low, and different networks are able to complete a better fit. The second is that the scenes are simple and the obstacles are fixed.

Table 1. Network structure experiment parameter table

Grouping	Numbering	Algorithm
1	1	TD3 four-ply
	2	TD3 three-ply
	3	DDPG four-ply
	4	DDPG three-ply

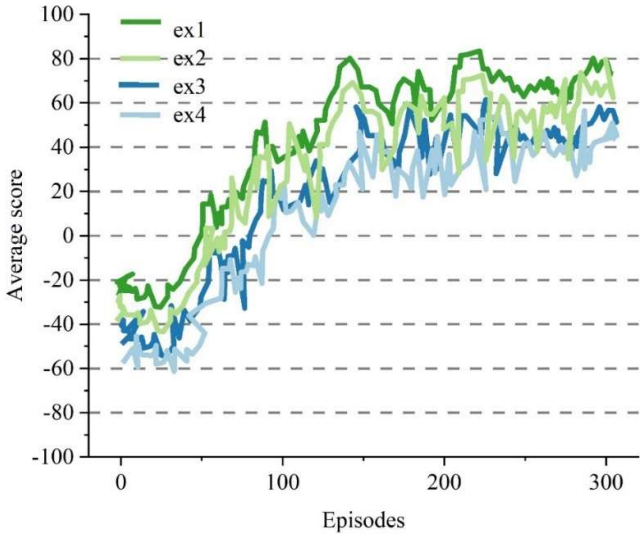

Fig. 6. Average score

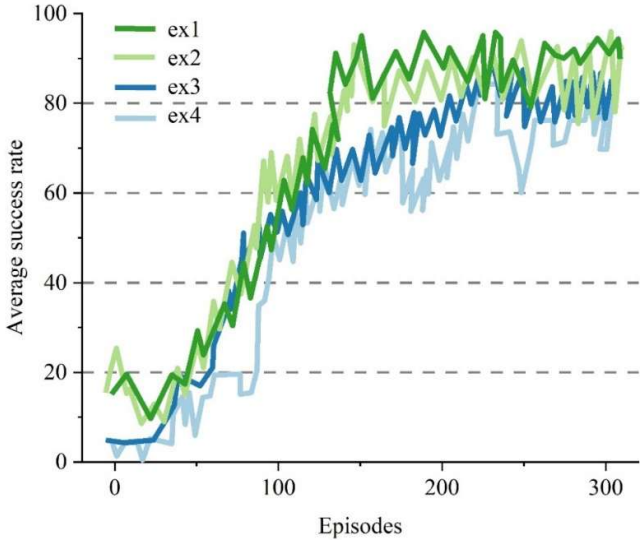

Fig. 7. Average success rate

This subsection tests the effect of different dimensions of the laser sensor on the results, with a view to achieving uniformity in training speed and effectiveness. A total of six experiments are divided into two groups, each group has three experiments, the experimental acceleration multiplier are 5 times speed, the number of the experiments and the parameter settings are shown in Table 2. Figure 8-Figure 11 show the changes of reward and success rate with the number of training acts during the training of the first and second group of experiments, respectively, and the data have been sliding averaged over 50 acts.

There is no observable effect gap between different dimensions of sensor information for both scene 1 and scene 3. The use of lower dimensional sensor information can reduce the amount of computation, but it should be noted that the scene is simpler and there is no small obstacle, if there is a small obstacle in the scene, the LIDAR dimension is too low will result in the inability to detect the small obstacle, so 20 dimensions were finally chosen as the laser data processing dimension.

Table 2. State space experiment parameter table

grouping	numbering	Laser data dimension	scene	Training schedule
1	1	80	1	300
	2	20	1	300
	3	5	1	300
2	1	80	3	500
	2	20	3	500
	3	5	3	500

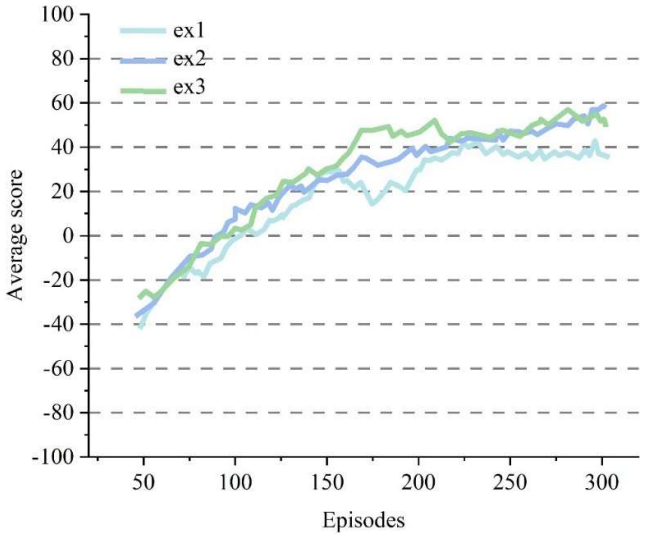

Fig. 8. Average score

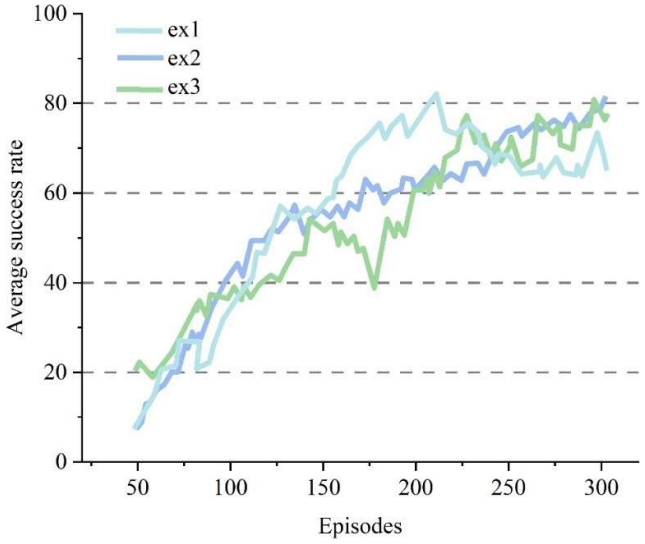

Fig. 9 Average success rate

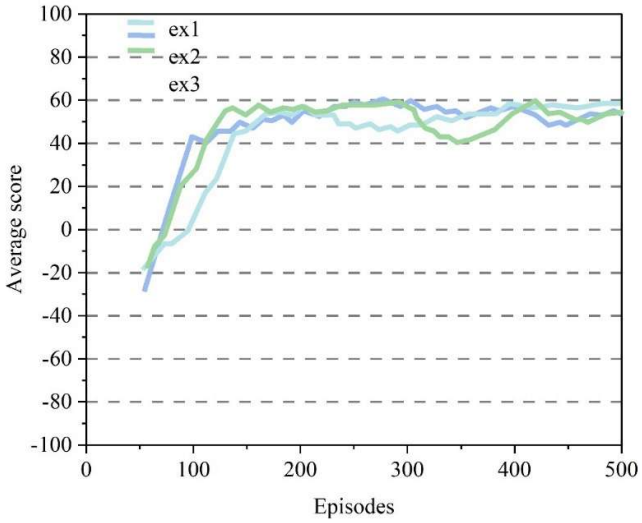

Fig. 10. Average score

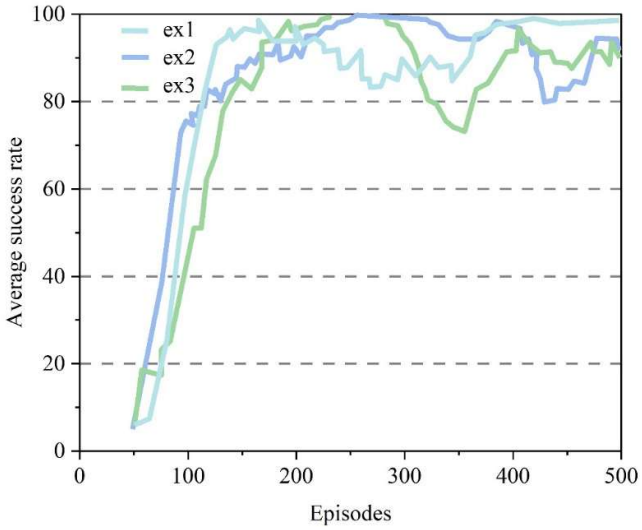

Fig. 11. Average success rate

6. Controller simulation experiment based on TD3 algorithm

In order to verify the performance of the trajectory tracking control algorithm proposed in this paper, MATLAB is used in this section to establish a mobile robot kinematics and dynamics simulation environment for the mobile robot to track the circular trajectory. The Simulink simulation system is established and S-functions are written based on the derived kinematic and dynamic models of the mobile robot and the designed kinematic and dynamic controllers, and the TD3 controller and PD controller are designed to track the circular trajectory, respectively. Fig. 12 and Fig. 13 show the simulation results of the trajectory tracking of the mobile robot under TD3 controller and PD controller, respectively. Under TD3 controller, after a period of time, the mobile robot can walk along the predetermined trajectory. Under PD controller, although the tracking trajectory of the mobile robot can enable the mobile robot to walk along the circular trajectory, but there is an error of about 20 cm in the radius of its operation. Fig. 14 and Fig. 15 show the circular trajectory tracking speed and position error curves, respectively, and the position error of the mobile robot tends to be close to 0 under the TD3 controller, while there is always an error in the X direction and Y direction of the mobile robot under the PD controller, with an error of up to 80 cm. Therefore, the trajectory tracking error of the TD3 controller designed in this paper is much smaller, and it has a higher control accuracy.

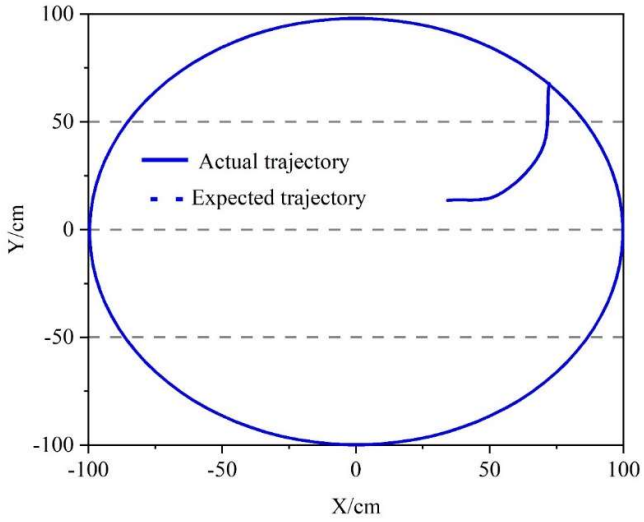

Fig. 12. TD3 controller trajectory tracking curve

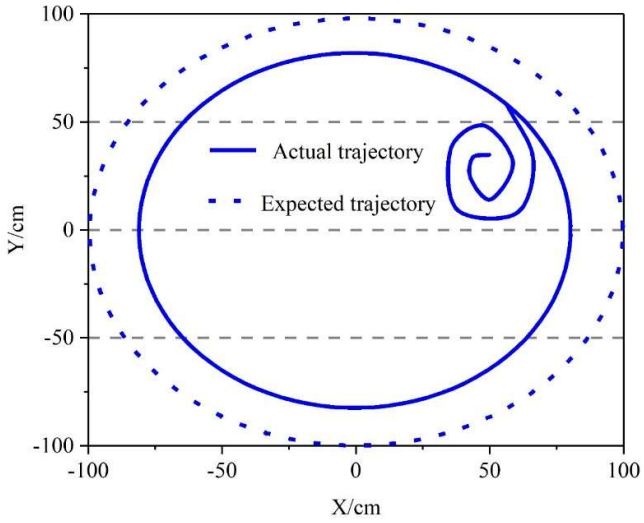

Fig. 13. PD controller trajectory tracking curve

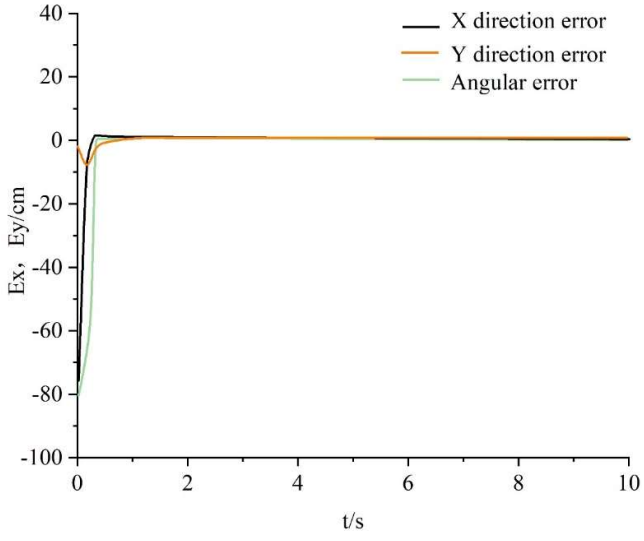

Fig. 14. The error curve of TD3 circular trajectory tracking position

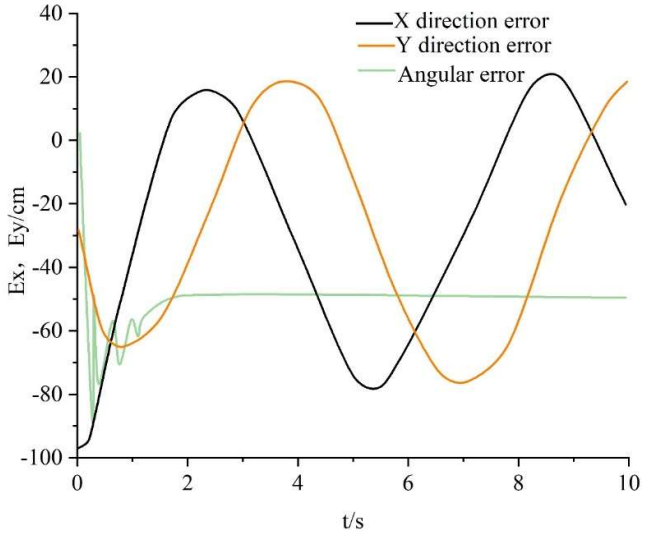

Fig. 15. The error curve of PD circular trajectory tracking position

Fig. 16 and Fig. 17 show the error curves of the left and right wheel speeds in circular trajectory tracking under the TD3 controller and the PD controller, respectively; the left and right wheel speeds in Fig. 16 converge faster than in Fig. 17, but the time of the tiny vibration is longer than that in Fig. 17, which is probably due to the online learning of the TD3 controller. Both controllers have the same final convergence speed with some error, which is generated by the motion controller with the same desired speed, indicating that the motion controller needs to be further optimized.

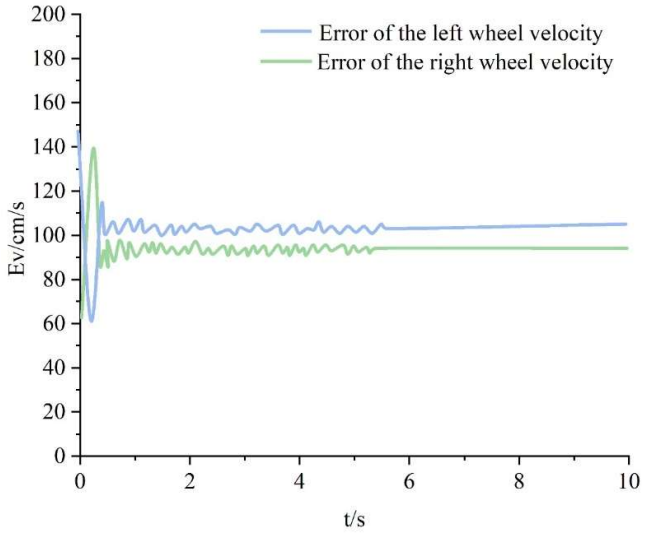

Fig. 16. The circular trajectory tracking velocity error curve controlled by TD3

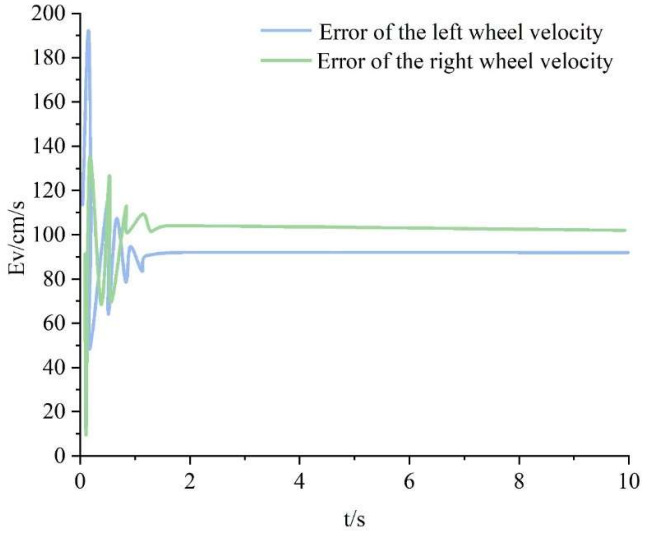

Fig. 17. The circular trajectory tracking velocity error curve controlled by PD

7. Conclusion

In this paper, based on deep reinforcement learning, a research on state space degradation and path tracking control optimization in robot navigation system is carried out. The main conclusions are as follows:

First, under the simple task scenario, the controllers with different network structures all achieve a success rate of 62% around 155 curtains, and the control effect is almost indistinguishable. When the input dimension is low, different network structures are able to fulfill the fitting task well, and there is no obvious control effect gap between sensors of different dimensions. And when 20-dimensional sensors are used, the computational amount of running is reduced. However, if the scene is more complex and contains small obstacles, the radar dimension is too low will lead to a decrease in the success rate of detecting small obstacles, so a more moderate 20-dimension is chosen as the dimension of laser data processing, which proves the necessity of the existence of the state space dimension reduction algorithm.

Second, the TD3 method is used in the robot navigation controller to realize online control, while the PD controller is designed as a comparison method of the TD3 control method in this paper. The position error of the navigating robot under this paper's controller tends to be close to 0, and the error of the X-direction and Y-direction of the mobile robot under the PD controller is up to 80 cm at the highest time.The simulation experiments have shown that the navigation tracking error of this paper's controller is much smaller, and it has a more accurate control ability.

References

[1] Gul, F., Rahiman, W., & Nazli Alhady, S. S. (2019). A comprehensive study for robot navigation techniques. *Cogent Engineering*, 6(1), 1632046.

[2] Crespo, J., Castillo, J. C., Mozos, O. M., & Barber, R. (2020). Semantic information for robot navigation: A survey. *Applied Sciences*, 10(2), 497.

[3] Nurmaini, S., & Tutuko, B. (2017). Intelligent robotics navigation system: Problems, methods, and algorithm. *International Journal of Electrical and Computer Engineering*, 7(6), 3711.

[4] Mao, W., Liu, H., Hao, W., Yang, F., & Liu, Z. (2022). Development of a combined orchard harvesting robot navigation system. *Remote Sensing*, 14(3), 675.

[5] Kumar, B. P., Suman, M. S., Samuel, D. P., Vadivel, V., & Ananth, C. (2015). Autonomous mobile robot navigation system. *International Journal of Advanced Research in Biology, Ecology, Science and Technology* (IJARBEST), 1(4), 15-19.

[6] Möller, R., Furnari, A., Battiato, S., Härmä, A., & Farinella, G. M. (2021). A survey on human-aware robot navigation. *Robotics and Autonomous Systems*, 145, 103837.

[7] Fauadi, M. H. F. M., Akmal, S., Ali, M. M., Anuar, N. I., Ramlan, S., Noor, A. Z. M., & Awang, N. (2018). Intelligent vision-based navigation system for mobile robot: A technological review. *Period. Eng. Nat. Sci*, 6(2), 47-57.

[8] Gao, X., Li, J., Fan, L., Zhou, Q., Yin, K., Wang, J., ... & Wang, Z. (2018). Review of wheeled mobile robots' navigation problems and application prospects in agriculture. *Ieee Access*, 6, 49248-49268.

[9] Pandey, A., Pandey, S., & Parhi, D. R. (2017). Mobile robot navigation and obstacle avoidance techniques: A review. *Int Rob Auto J*, 2(3), 00022.

[10] Ren, J., Wu, T., Zhou, X., Yang, C., Sun, J., Li, M., ... & Zhang, A. (2022). SLAM, path planning algorithm and application research of an indoor substation wheeled robot navigation system. *Electronics*, 11(12), 1838.

[11] Verbitsky, N. S., Chepin, E. V., & Gridnev, A. A. (2018). Experimental studies of a convolutional neural network for application in the navigation system of a mobile robot. *Procedia computer science*, 145, 611-616.

[12] Atiyah, A. N., Adzhar, N., & Jaini, N. I. (2021, July). An overview: On path planning optimization criteria and mobile robot navigation. *In Journal of Physics: Conference Series* (Vol. 1988, No. 1, p. 012036). IOP Publishing.

[13] Jalali, S. M. J., Khosravi, A., Kebria, P. M., Hedjam, R., & Nahavandi, S. (2019, October). Autonomous robot navigation system using the evolutionary multi-verse optimizer algorithm. *In 2019 IEEE international conference on systems, man and cybernetics* (SMC) (pp. 1221-1226). IEEE.

[14] Boufera, F., Debbat, F., Monmarché, N., Slimane, M., & Khelfi, M. F. (2018). Fuzzy inference system optimization by evolutionary approach for mobile robot navigation. *International Journal of Intelligent Systems and Applications*, 15(2), 85.

[15] Lazreg, M., & Benamrane, N. (2022). Hybrid system for optimizing the robot mobile navigation using ANFIS and PSO. *Robotics and Autonomous Systems*, 153, 104114.

[16] Yang, H., Qi, J., Miao, Y., Sun, H., & Li, J. (2018). A new robot navigation algorithm based on a double-layer ant algorithm and trajectory optimization. *IEEE Transactions on Industrial Electronics*, 66(11), 8557-8566.

[17] Sellers, T., Lei, T., Jan, G. E., Wang, Y., & Luo, C. (2022, June). Multi-objective optimization robot navigation through a graph-driven PSO mechanism. *In International Conference on Sensing and Imaging* (pp. 66-77). Cham: Springer International Publishing.

[18] Meléndez, A., Castillo, O., Valdez, F., Soria, J., & Garcia, M. (2013). Optimal design of the fuzzy navigation system for a mobile robot using evolutionary algorithms. *International Journal of Advanced Robotic Systems*, 10(2), 139.

[19] Hamza, E. K., Salman, K. D., & Jaafar, S. N. (2023). Wireless Sensor Network for Robot Navigation. *In Mobile Robot: Motion Control and Path Planning* (pp. 643-670). Cham: Springer International Publishing.

[20] Muhammad Ikram & Robert Sroufe. (2025). A novel sequential block path planning method for 3D unmanned aerial vehicle routing in sustainable supply chains. Supply Chain Analytics100094-100094.

[21] Di Yuan & Yue Wang. (2025). Sustainable supply chain management: A green computing approach using deep Q-networks. Sustainable Computing: Informatics and Systems101063-101063.

[22] Jiachen Yang,Jiankun Peng,Quanwei Zhang,Weiqi Chen & Chunye Ma. (2025). Monocular vision approach for Soft Actor-Critic based car-following strategy in adaptive cruise control. Expert Systems With Applications125999-125999.